NASA/TM—2014-216663

Tool for Turbine Engine Closed-Loop Transient Analysis (TTECTrA) Users' Guide

Jeffrey T. Csank
Glenn Research Center, Cleveland, Ohio

Alicia M. Zinnecker
N&R Engineering and Management Services, Inc., Parma Heights, Ohio

June 2014

NASA STI Program . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI Program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI program provides access to the NASA Aeronautics and Space Database and its public interface, the NASA Technical Reports Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or cosponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include creating custom thesauri, building customized databases, organizing and publishing research results.

For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at *http://www.sti.nasa.gov*

- E-mail your question to *help@sti.nasa.gov*

- Fax your question to the NASA STI Information Desk at 443–757–5803

- Phone the NASA STI Information Desk at 443–757–5802

- Write to:
  STI Information Desk
  NASA Center for AeroSpace Information
  7115 Standard Drive
  Hanover, MD 21076–1320

NASA/TM—2014-216663

Tool for Turbine Engine Closed-Loop Transient Analysis (TTECTrA) Users' Guide

Jeffrey T. Csank
Glenn Research Center, Cleveland, Ohio

Alicia M. Zinnecker
N&R Engineering and Management Services, Inc., Parma Heights, Ohio

National Aeronautics and
Space Administration

Glenn Research Center
Cleveland, Ohio 44135

June 2014

Level of Review: This material has been technically reviewed by technical management.

Available from

NASA Center for Aerospace Information
7115 Standard Drive
Hanover, MD 21076–1320

National Technical Information Service
5301 Shawnee Road
Alexandria, VA 22312

Available electronically at http://www.sti.nasa.gov

Contents

List of Tables

List of Figures

Tool for Turbine Engine Closed-Loop Transient Analysis (TTECTrA) Users' Guide

Jeffrey T. Csank
National Aeronautics and Space Administration
Glenn Research Center
Cleveland, Ohio 44135

Alicia M. Zinnecker
N&R Engineering and Management Services, Inc.
Parma Heights, Ohio 44130

Abstract

The tool for turbine engine closed-loop transient analysis (TTECTrA) is a semi-automated control design tool for subsonic aircraft engine simulations. At a specific flight condition, TTECTrA produces a basic controller designed to meet user-defined goals and containing only the fundamental limiters that affect the transient performance of the engine. The purpose of this tool is to provide the user a preliminary estimate of the transient performance of an engine model without the need to design a full nonlinear controller.

1.0 Introduction

TTECTrA has been developed in the MATLAB/Simulink (The Mathworks Inc.) environment which allows users to access a standard library of functions and to add on toolboxes such as the Control System Toolbox (The Mathworks Inc.), which can be used to simplify the control design process (and is required to use TTECTrA). This user's guide is written assuming the user is familiar with MATLAB and Simulink.

TTECTrA consists of custom MATLAB functions written to perform control design calculations and to interact with the user's Simulink engine model. The tool also contains the *TTECTrA Simulink Block* (in the *TTECTrA_block.mdl* file), which implements a scheduled proportional integral (PI) controller with the designed setpoints, gains, and limiters and supplies the fuel flow input to the user's engine model. More information regarding the integration of TTECTrA with an engine model is contained in Section 2.0.

TTECTrA has been tested and verified using MATLAB Version 8.0 (R2012b), Simulink Version 8.0 (R2012b), and the Control System Toolbox Version 9.4 (R2012b). The tool integrates with an engine model written in Simulink and requires a piecewise linear state space model of the engine to be available. Note that, throughout this paper, MATLAB commands, functions, variables, files, and subsystems will be *italicized.*

2.0 Installation and Model Setup

2.1 Software Requirements For Using TTECTrA

The TTECTrA tool integrates with an engine model that is compatible with MATLAB/Simulink and provides an interface for designing and implementing a gain scheduled PI controller with acceleration and deceleration limiters. The following requirements must be met in order to use TTECTrA:

1. MATLAB/Simulink R2012b, with the Control System Toolbox installed (TTECTrA may work with other versions of MATLAB/Simulink, but has been developed and tested with R2012b)
2. An engine model implemented in Simulink that provides the following outputs (for feedback):
   a. The control variable, such as fan speed (*Nf*) or engine pressure ratio (*EPR*)
   b. Thrust (corrected or uncorrected)

c. Core speed (*Nc*)
d. Measurements of (total) pressure and temperature at stations 2 and 25 (for correcting feedback signals), and measurement of static pressure at station 3.

The engine model should take, as input from TTECTrA, a fuel flow input; other inputs, such as variable geometry, should be included as part of the user's engine model because the controller designed by TTECTrA only handles fuel flow.

A piecewise-linear form of the engine model, where the linear models have been found for a set of thrust breakpoints and placed in the structure described in Appendix B.—Linear Model Input Requirements. (These models will be used for designing the setpoint controller gains.)

2.2 Installation

The TTECTRA code consists of a Simulink block and a folder of custom MATLAB scripts and functions which are called during the execution of TTECTrA. To install TTECTrA for use with an engine model:

1. Right click on the zip package and choose Extract All. This will start the extraction wizard.
2. Select "Next" to bring up the Select a Destination screen.
3. Select "Browse" to open the file browser window and navigate to the folder that contains the engine model that will interact with TTECTrA. Select OK to return to the extraction wizard.
4. Select "Next" to extract all files to the desired location.
5. Select "Finish" to close the extraction wizard.

2.3 Integrating TTECTrA With an Engine Model

The TTECTrA tool utilizes features in MATLAB and Simulink to design and implement a scheduled PI controller, with acceleration and deceleration limiters, for an engine model built in Simulink. The user's model must be integrated with this tool prior to proceeding with control design; the steps for doing so are presented in this section, following a description of the architecture of the controller implemented by the TTECTrA block.

2.3.1 Controller Architecture

The general architecture of the controller designed and implemented by the TTECTrA tool is shown in Figure 1, where the details of each block are omitted for simplicity. A switch is used to select the fuel flow calculated using one of three possible control methods (identified by the "loop selection" flag):

1. Closed-loop control with the controller designed using TTECTrA to calculate the setpoint map, controller PI gains, and acceleration and deceleration limiters
2. Closed-loop control where the system is driven to a demanded thrust value as fast as possible; this controller is only used for setpoint calculation
3. Open-loop control where a fuel flow profile is defined; this method may be used for setpoint calculation, and is also used for calculating the limiters

The two feedback signals (control variable and thrust) are provided by the engine model, along with (corrected) core speed and *Ps3* (which are used by the "acceleration" and "deceleration" logic blocks respectively). The fuel flow command is provided to the engine to close the loop with the TTECTrA block, but it is important to note that this is *not* a closed-loop *simulation* if the fuel flow demand input is selected.

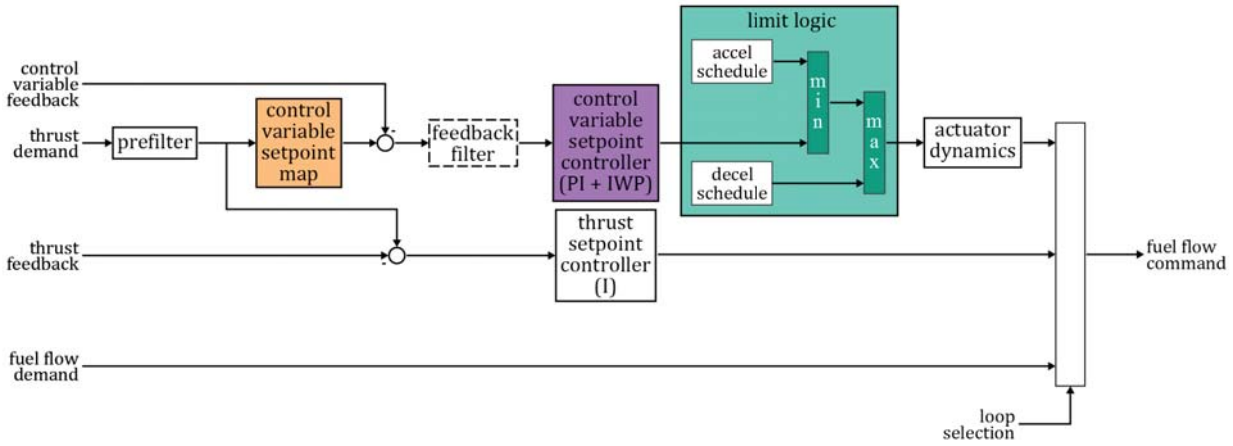

Figure 1.—Schematic of the controller implemented by the TTECTrA Simulink block; parameters in the highlighted blocks are designed by TTECTrA.

Even though TTECTrA includes a simple controller to drive the system to a reference thrust, this is not a realistic controller because there are no "thrust sensors" in an actual engine to provide the feedback data (this data is available in an engine model for analysis purposes). Instead, a measureable quantity that is related to thrust (typically fan speed or *EPR*) is used as the feedback to the controller; this model-dependent measurement is referred to as the "control variable" throughout this document. The setpoint controller acts to drive the control variable to a reference value mapped from the demanded thrust using a relationship calculated for the specific engine model; consequently, if the control variable tracks the demand, it is expected that the thrust produced by the engine also tracks the demanded thrust. The tracking response is determined by how the proportional and integral gains of the controller are tuned and scheduled and by how tight the defined limiters for acceleration and deceleration are.

The acceleration limiter is designed to prevent the high-pressure compressor from surge during engine acceleration. The core acceleration is compared to an acceleration limit that is dependent on the core speed; fuel flow is restricted if the acceleration is too high. The core acceleration limit as a function of corrected core speed is often called an acceleration schedule. The acceleration schedule is found by applying fuel flow profiles transitioning from an idle fuel flow to a takeoff fuel flow at different rates until the minimum high-pressure compressor surge margin requirement is met. The acceleration limiter uses a PI controller, with integral wind-up protection, to produce a fuel flow command designed to drive the engine acceleration to the acceleration limit.

The deceleration limit is found by applying fuel flow transitions from a takeoff fuel flow to an idle fuel flow and finding the Wf/Ps3 value which would preserve the minimum acceptable low-pressure compressor surge margin. The fuel flow command produced by this limiter is calculated by multiplying the combustor pressure (Ps3) by the Wf/Ps3 limit.

The fuel flow command provided to the "actuator dynamics" block and, ultimately, to the engine, is determined by first selecting the minimum of the fuel flow from the setpoint controller and that from the acceleration limiter. This fuel flow is then compared to the command from the deceleration limiter and the largest of these commands is selected. In this way, the fuel flow provided to the engine represents the controller that is operating closest to its respective setpoint. For more information regarding the design of an aircraft engine controller, the reader is referred to References 1 to 3.

Running TTECTrA allows the user to design parameters for implementation of the three highlighted blocks in the figure: the setpoint map (relationship between thrust and control variable), the gain schedules for the PI controller, and the limiter logic that depends on the acceleration schedule and the lower limit on *Wf/Ps3* (deceleration schedule). A set of graphical user interfaces (GUIs) has been created to guide the user through these design steps, which will be discussed in more detail in Section 3.0. The GUIs are implemented by a set of MATLAB scripts that take advantage of built-in functions that can run Simulink models from the command line; this enables interaction between MATLAB and Simulink while TTECTrA is running, which is discussed in the next section.

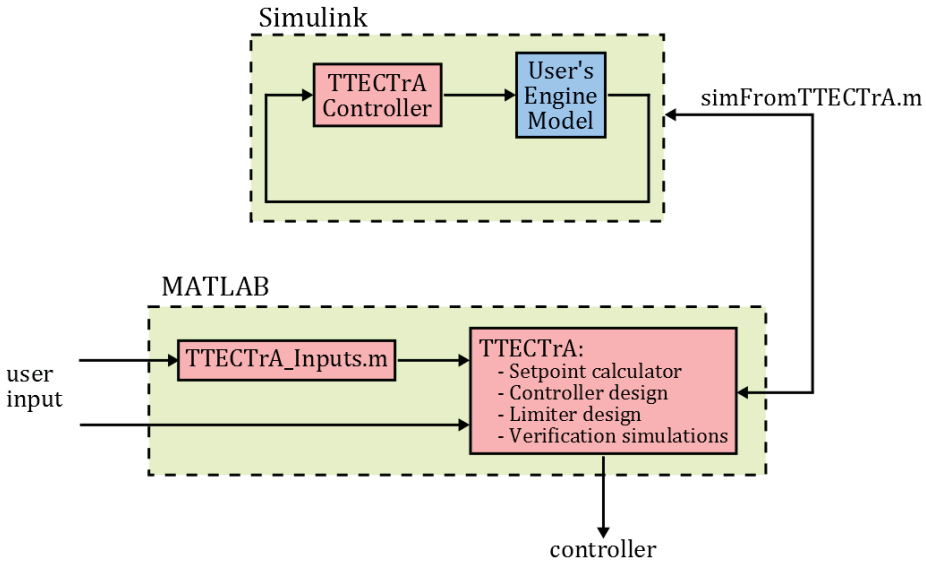

Figure 2.—Representation of how TTECTrA interacts with MATLAB and Simulink for designing and verifying a controller.

2.3.2 Interaction Between TTECTrA and MATLAB/Simulink

The TTECTrA tool has been designed to interact with MATLAB (through custom Matlab scripts, or .m files) and also with Simulink (though the included Simulink block, which implements the TTECTrA Controller block with the architecture discussed above). The framework of these interactions is represented in Figure 2, where the blocks highlighted in pink are implemented by TTECTrA and the block highlighted in blue is user-provided. The MATLAB code interacts with the Simulink model through calls to the MATLAB function *sim* in the *simFromTTECTrA.m* file, which is used in each of the four steps of design and verification, as listed in the "TTECTrA" block in Figure 2. In this way, it is only necessary to run the main file (*TTECTrA.m*) from the MATLAB command line to perform all design operations and run the test simulations.

The user interfaces with the tool in two ways: by adding the TTECTrA controller block to their Simulink model file, which contains their engine model, and by providing parameters for the setpoint, controller, and limiter that TTECTrA will use in designing the controller. The steps for setting up the model and input file will be described in detail in the following sections.

2.3.3 Setting Up the Simulink Engine Model

Before using TTECTrA, the *TTECTrA Simulink Block* should be integrated with the user's Simulink engine model following these steps:

1. Copy and paste the *TTECTrA Simulink Block* (found in the *TTECTrA_block.mdl* file) into the Simulink model containing the engine.
2. Connect the inputs of the *TTECTrA Simulink Block*, listed in Table 1, to the appropriate outputs of the engine model.
3. Connect the output of the *TTECTrA Simulink Block*, listed in Table 2, to the fuel flow input of the engine model. Note that the fuel flow actuator dynamics are modeled inside this block as a first-order filter, with a user-specified bandwidth. An example of the *TTECTrA Simulink Block* integrated with an example engine model is shown in Figure 3.

TABLE 1.—INPUTS TO THE *TTECTrA SIMULINK BLOCK*

Variable Name	Description
Fdbk	Control variable output of the engine (feedback to the controller)
FnR	Corrected net thrust of the engine
Time	Simulation time
NcR25	High pressure spool rotor speed corrected to station 25
Ps3	High pressure compressor static discharge pressure

TABLE 2.—OUTPUTS FROM THE *TTECTrA SIMULINK BLOCK*

Variable Name	Description
Wf_engine	Fuel flow to the engine; includes fuel flow actuator dynamics

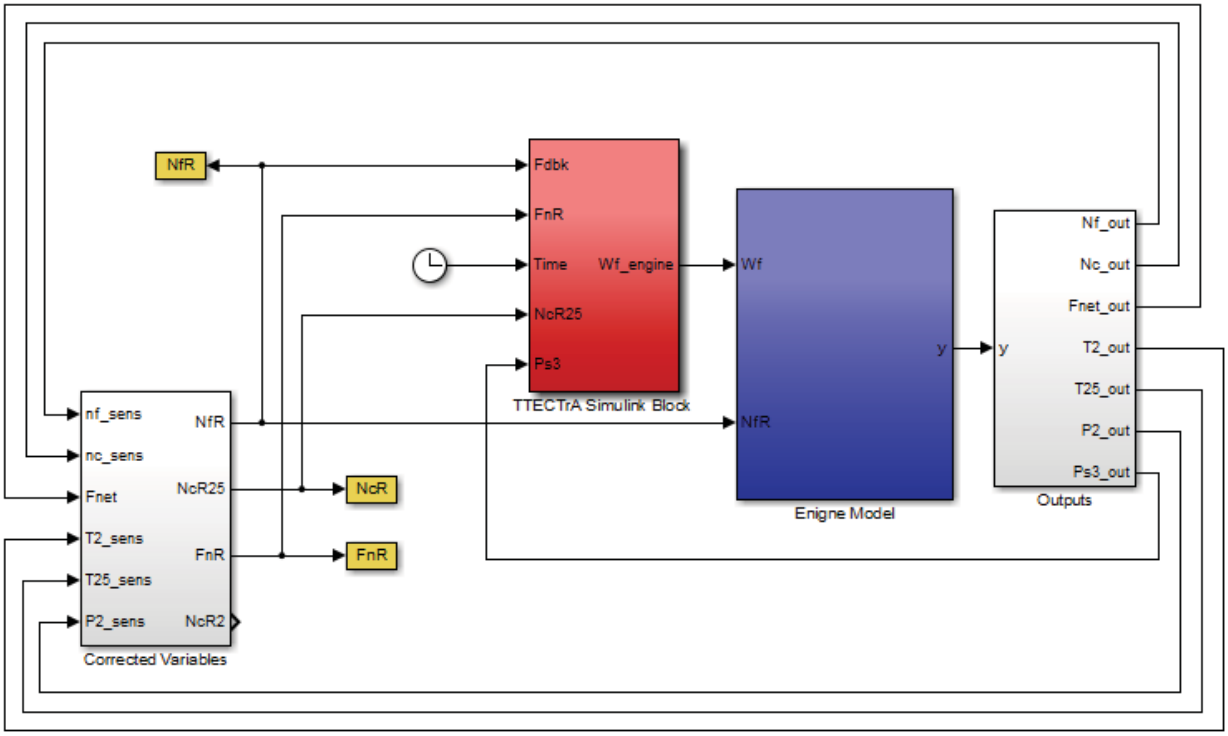

Figure 3.—The TTECTrA Simulink block integrated with an example engine model.

2.3.4 Setting Up the TTECTrA MATLAB File

The final part of the setup is to integrate the Simulink model with the TTECTrA MATLAB code. This requires the user to modify the *simFromTTECTrA.m* MATLAB file, which uses three structured variables to communicate between TTECTrA and the Simulink model: *inputs*, *outputs*, and *DWS*. The *inputs* variable (Table 3) contains data from TTECTrA for use by the Simulink model simulation. The *outputs* variable (Table 4) is the collection of the workspace outputs from the Simulink model. The *DWS* variable (Table 5) contains all the information required by the *TTECTrA Simulink block*. The file *simFromTTECTrA.m*, which runs the simulation, is separated into two sections: model-specific workspace setup and model execution setup. An external file, *setup_TTECTrA_block.m*, is called to create the *DWS* variable used by the controller block and requires no modification by the user.

To link the TTECTrA MATLAB code with the full Simulink engine model, the user must:

1. Set the MATLAB workspace up for simulation of their model (with a given command or set of commands) in the "Model-specific workspace setup" section of *simFromTTECTrA.m*.
2. Write the required Simulink outputs to TTECTrA by modifying the "Model execution setup" section of *simFromTTECTrA.m*.

The following sections contain instructions for making these modifications, using an example application of TTECTrA with a piecewise-linear engine model.

2.3.4.1 Model-Specific Workspace Setup

The model-specific workspace setup section of the *simFromTTECTrA.m* file should contain all the code necessary to setup the MATLAB workspace for simulation; as such, this section of the code requires the most user modification for TTECTrA to run successfully.

The user-provided code should perform the following functions:

1. Unpack the flight condition from the *.in* field of *inputs* (*inputs.in*) (shown in Table 3). The flight condition is defined by three scalar values: altitude (*alt*), Mach number (*MN*), and deviation of ambient temperature from standard-day temperature (*dTamb*).

TABLE 3.—FIELDS OF THE *inputs* STRUCTURE VARIABLE ARGUMENT TO *simFromTTECTrA.m*

Field	Field Name	Description
in	t_vec	Time (vector)
	Alt	Altitude (scalar)
	MN	Mach number (scalar)
	dTamb	Temperature deviation from standard day condition (scalar)
	simTime	Length time for simulation to run
	simFileName	File name of user's engine model
	loop	Controller switch (1=control variable, 2=solver for thrust, 3=open loop)
	FT_dmd	Thrust demand (vector or scalar (if constant thrust)); required only for in.loop = 1 or in.loop = 2
	wf_vec	Fuel flow demand (vector or scalar (if constant fuel flow)); required only for in.loop = 3
	Fdbk_flag	Flag for feedback filter (= 1)
SPcalc	wf_rng	Fuel flow range for engine model ([wfmin wfmax])
	idle	Idle thrust
	takeoff	max takeoff thrust
	bkpt	breakpoints
controller	FdbkFilterBw	Bandwidth for feedback filter if >0, else no filter used
	PreFilterBW	Bandwidth for prefilter (filters the thrust command or setpoint)
actuator	wf_bw	Bandwidth for first-order filter modeling fuel actuator dynamics
SMLimit	Accel	Desired minimum surge margin during an acceleration
	Decel	Desired minimum surge margin during a deceleration
gains	Kp	Proportional controller gain (vector, scheduled by control variable)
	Ki	Integral controller gain (vector, scheduled by control variable)
	Fdbk	Control variable breakpoints (vector, for gain scheduling)
SP	FT_bkpt	Thrust breakpoints (vector)
	SP	Control variable setpoints (vector, scheduled by thrust)
Limiter	NcR25_sched	Corrected core speed at station 25 (vector, for acceleration schedule)
	Ncdot_sched	Core acceleration limit (vector, scheduled by corrected core speed)
	LPC_Limiter	Wf/Ps3 limit (scalar)

If, for example, the workspace is setup using a function that requires a structure containing these environmental variables, the following code may be used to unpack those values:

```
% setup vectors defining altitude, Mach number, dTamb (and time)
in.t_vec = inputs.in.t_vec;
in.alt   = inputs.in.alt;
in.MN    = inputs.in.MN;
in.dTamb = inputs.in.dTamb;
```

(Additional formatting may be necessary if, for instance, these conditions should be provided to the setup function as vectors instead of scalars.)

2. Assign the default simulation name to the *inputs.in.simFileName* variable to ensure that the simulation executes:

```
if ~isfield(inputs,'in') || ~isfield(inputs.in,'simFileName') ...
                         || isempty(inputs.in.simFileName)
    inputs.in.simFileName = 'filename.ext';    % modify this
end
```

3. Add any additional MATLAB code required to setup the workspace and trim the model to the initial conditions. This code may be included directly in the file, or may be contained in external function created to execute any of the following steps:

   a. Adding folders containing functions or data files needed during the simulation to the current MATLAB path
   b. Loading data, or creating variables, needed by the engine model (e.g. compressor maps, lookup table data, sampling time)
   c. Trimming the model to the initial fuel flow or thrust demand in order to define the initial condition of the simulation

   For a model that places all data needed for a simulation into a single workspace variable and uses a lookup table for trimming the model, this can be done by the following code:

```
% trim model to initial thrust demand, if closed-loop simulation
if isfield(inputs.in,'FT_dmd');
    wf_0 = trim_model(inputs.in.FT_dmd(1),9);
else
    wf_0 = inputs.in.wf_vec(1);
end

% add paths, find initial conditions, create MWS
MWS=setup_workspace(in.t_vec,wf_0);
```

   Here, *trim_model* performs the table lookup for the initial fuel flow and *setup_workspace* performs the remaining setup tasks from the above list. (A more complex model may use a steady-state solver in place of a lookup table to trim the model.)

4. Assign the model sampling time, initial fuel flow, and initial core speed to the *in* field of the *DWS* structured variable, along with the initial conditions for pressure and temperature at station 2:

```
DWS.in.Ts_cont = MWS.Ts;          % model sampling time
DWS.in.Wf_zro  = MWS.IC.Wf_0;     % initial fuel flow
DWS.in.Nc_zro  = MWS.IC.Nc_0;     % initial core speed
DWS.in.P2      = MWS.IC.P2_0;     % initial P2, used for correction
DWS.in.T2      = MWS.IC.T2_0;     % initial T2, used for correction
```

2.3.4.2 Model Execution Setup

The model execution setup section of the *simFromTTECTrA.m* file contains the MATLAB commands that run the model simulation and place the results in the *outputs* variable, which contains the fields listed in Table 4. The model is simulated using the function *sim*, called with the output argument *y* (to which simulation outputs are returned); each individual output can be accessed using the *get* command and specifying the variable name as it appears in the Simulink model.

For example, if pressure at station 2 is written to the workspace variable *P2*, the field 'P2' of *outputs* would be assigned using:

```
outputs.P2      = y.get('P2');
```

This may be done for each field of *outputs* listed in Table 4, which are required for use during the control design process.

Because *get* accepts a single argument (the variable name), additional manipulation of the output variables must be done external to retrieval of results. This may be exemplified by the assignment of the field for corrected thrust, 'Fnet,' in the case that uncorrected thrust is written to the workspace from the model. Assuming uncorrected thrust is written to the workspace variable *Fnet* by the model, the following code cannot be used:

```
outputs.Fnet    = y.get('Fnet./P2')/14.696;    % this doesn't work
```

TABLE 4.—FIELDS OF THE *OUTPUTS* STRUCTURE VARIABLE RETURNED BY *simFromTTECTrA.m*

Field Name	Description
t	Time vector
P2	Inlet pressure
Fnet	Corrected net thrust
Wf_vec	Fuel flow input
T25	Temperature at station 25
Nc	Core (or high spool) speed
NcR25	Corrected core (or high spool) speed (used for acceleration schedule)
Nc_dot	Core acceleration (used for acceleration schedule)
HPC_SM	High pressure compressor surge margin (used for acceleration schedule)
LPC_SM	Low pressure compressor surge margin (used for Wf/Ps3 limiter)
CV_fdbk	Controlled variable output
CV_dmd	Control variable setpoint or demand
FT_dmd	Thrust setpoint or demand
Wf_dmd	Fuel flow input or demand
Ps3	High pressure compressor static pressure (used for Wf/Ps3 limiter)

Instead, it is necessary to retrieve the arrays *Fnet* and *P2* separately and do the calculation using:

```
outputs.Fnet    = y.get('Fnet')./(outputs.P2/14.696);
```

The variables necessary for TTECTrA to function are indicated in the *simFromTTECTrA.m* file, and listed in Table 4: 10 output fields assigned from outputs of the engine model and four from outputs of the *TTECTrA Simulink block*. Additional outputs, such as *Ps3*, may also be returned by retrieving the workspace variable and storing it in a corresponding field of *outputs*.

2.3.4.3 TTECTrA Simulink Block Setup

Prior to running the simulation, the *simFromTTECTrA.m* file calls the function *setup_TTECTrA_block.m* to create the *DWS* variable (used by the TTECTrA Simulink block) from data in the *inputs* variable. The fields contained in *DWS* are shown in Table 5. The user should not have to modify this function, as it pertains to the controller designed using TTECTrA and not to a specific engine model.

TABLE 5.—THE *DWS* VARIABLE STRUCTURE

Field	Field Names	Description
in	loop	Control/Feedback indicator
	t_vec	Time vector
	wf_vec	Fuel flow input (default values unless loop=3)
	FT_dmd	Thrust demand (default values unless loop=1 or loop=2)
	Fdbk_Flag	Flag to enable/disable filter in feedback loop
	Ts_cont	Simulation sample time (from user's model)
	Wf_zro	Initial fuel flow (from user's model)
	Nc_zro	Initial core speed (from user's model)
TTECTrA_controller	Fdbk_num_Z	Discrete feedback filter numerator
	Fdbk_den_Z	Discrete feedback filter denominator
	PreFilterBW	Bandwidth of prefilter on thrust command (Hz)
	Fdbk	Feedback breakpoints (for controller gain lookup tables)
	P_gain	Proportional gains (lookup table data)
	I_gain	Integral gains (lookup table data)
	IWUP	Integral Wind-up Protection gain
TTECTrA_Wf	bandwidth	Bandwidth of first-order filter modeling fuel flow dynamics
TTECTrA_Limiter	Nc_sched	Corrected core speed breakpoints (for acceleration schedule lookup table)
	Ncdot_sched	Core acceleration limits (lookup table data)
	WfPs3Limit	Deceleration protection limit on Wf/Ps3
	Kp_accel	Proportional gain for the acceleration limiter
	Ki_accel	Integral gain for the acceleration limiter
	IWUP_accel	Integral Wind-up Protection for the acceleration limiter
	accel_num	Acceleration filter numerator
	decel_num	Acceleration filter denominator
TTECTrA_setpoints	FT_bkpt	Thrust feedback breakpoints (for setpoint lookup table)
	SP	Controlled variable setpoints (lookup table data)

3.0 TTECTrA Operation

This section focuses on setting up and operating the Tool for Turbine Engine Closed-loop Transient Analysis, which can be done once it has been integrated with the user's nonlinear engine model as described in Section 2.0. An example application based off of the Commercial Modular Aero-Propulsion System Simulation 40,000 (C-MAPSS40k) (Ref. 4) has been included with TTECTrA (*TTECTrA_example.mdl*). As the tool is discussed, it may be helpful to follow along using this example, which is a piecewise-linear version of the C-MAPSS40k engine model. In addition, a set of linear models developed at sea-level static conditions (0 ft and 0 Mach number) are included in the file *LM_PWL.mat*

for use in developing a controller; a pre-design controller can be found in the file *TTECTrA_example_design.mat*. Before running TTECTrA with this model, the *make_file.m* file (found in example_model/MEX/C_code) must be run to create the supporting code for the simulation.

A typical control design using TTECTrA involves three main steps: calculating the setpoint map, finding the controller gain schedules, and calculating the acceleration and deceleration limiters. After TTECTrA has been started, the steady-state mapping between corrected thrust and the control variable setpoint is calculated. This relationship is dependent on the engine model and requires that the user connects the appropriate feedback signal to the *Fdbk* input of the *TTECTrA Simulink block* in the Simulink model before performing the calculation. Next, the tool calculates the controller gains for each model composing the provided piecewise-linear model of the engine; these controller gains will be scheduled (based on control variable) for implementation in the setpoint controller. The final set of calculations addresses the need for implementing transient limiters to protect the engine from surge; acceleration and deceleration schedules can be found to ensure the user-specified minimum surge margins for the high- and low-pressure compressors (HPC and LPC) are not violated during periods of high engine demand. The TTECTrA controller only considers these two limiters, but it is possible to expand the limit logic to include additional constraints, such as core speed or *Ps3*.

After the controller has been designed, two simulations will be run to test the functionality of the controller: a simulation with small changes in thrust demand (to test the setpoint controller) and a simulation with large changes in thrust demand (to test the limiters). The results for the control design and verification for the example model will be presented here along with discussion of each step of the design process using TTECTrA.

3.1 Getting Started

Before operating TTECTrA, the user has the option to specify default values and preferences for the parameters listed in Table 6 in the file *TTECTrA_Inputs.m*. The values in this file are loaded by TTECTrA and recalled when the GUI is started, but may be changed during the design process if necessary.

TABLE 6.—USER DEFAULT INPUTS FROM *TTECTrA_Inputs.m* FILE, WHICH ARE LOADED INTO THE GUI

SubField	Field Name	Description
in	alt	Altitude (scalar)
	MN	Mach number (scalar)
	dTamb	Ambient temperature deviation from standard day (scalar)
	simTime	Length of the simulation (scalar)
	simFileName	File name (and extension) of the user's Simulink engine model with the *TTECTrA Simulink Block* controller
SPcalc	wf_rng	Fuel flow range ([min max])
	idle	Idle corrected thrust
	takeoff	Takeoff corrected thrust
	bkpt	If scalar, specifies the number of linearly spaced thrust values from idle to takeoff for calculating setpoints If vector, defines specific thrust breakpoints for calculating setpoints
controller	LMFileName	Name of .mat file which contains linear model
	lmVar	Name of variable containing the linear model data
	FdbkFilterBW	Feedback filter bandwidth if > 0, otherwise no filter is used
	PreFilterBW	Prefilter bandwidth
	CVoutput	Element of the linear model output vector (y_i) corresponding to the controlled variable
	Wfinput	Element of the linear model input vector (u_i) corresponding to fuel flow
	bandwidth	Default bandwidth for tuning the controller
	phasemargin	Default phase margin for tuning the controller
actuator	wf_bw	Bandwidth of filter modeling fuel flow dynamics
SMlimit	Accel	Minimum allowed surge margin during acceleration (for limiter design)
	Decel	Minimum allowed surge margin during deceleration (for limiter design)

3.2 GUI Operation

To begin the control design using TTECTrA, run the *TTECTrA.m* file. The dialog box shown in Figure 4 will appear, asking if a previously-saved controller should be loaded. To load a previously-designed controller (with the option of full or partial redesign of the loaded controller), select "yes" and proceed as follows when the Load Controller Data GUI of Figure 5 appears. Otherwise, select "no" to bring up the Setpoint Calculator GUI (Go to "Setpoint Calculator" section).

1. Press the "Choose File to Load" button to browse to and select the appropriate file. If the file is not on the current MATLAB path, the location of the file will be added to the path to ensure it can be loaded successfully. (The file *TTECTrA_example_design.mat*, located in the *example_model* folder, contains data for a controller designed for the example model.)
2. Verify that the correct file name appears in the text box, and then press the "Load Data" button to load the controller. MATLAB will issue a warning if any required fields are missing from the data; these fields may be assigned later by TTECTrA. An error will be encountered if there is a problem loading the appropriate data from the file (e.g. the file name is incorrect, or the file does not contain a variable named *inputs*).
3. The Controller Redesign GUI in Figure 6 will appear, asking which parts of the controller the user would like to redesign, if any; check the appropriate boxes then click Continue. If the loaded controller is missing any of the three parts required by TTECTrA, the corresponding check boxes will be marked as shown in Figure 6, forcing the controller design to complete before simulation.
4. If no part of the controller is to be redesigned, the model will be simulated immediately after the controller data is loaded (Go to the "Verify and Execute Simulation" section). Otherwise, TTECTrA will proceed with the specified controller design steps.

Figure 4.—Dialog box asking if the user would like to load previously designed controller data.

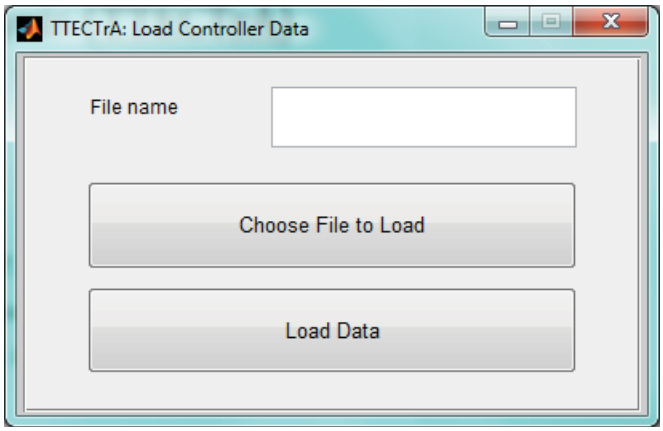

Figure 5.—Load Controller Data GUI.

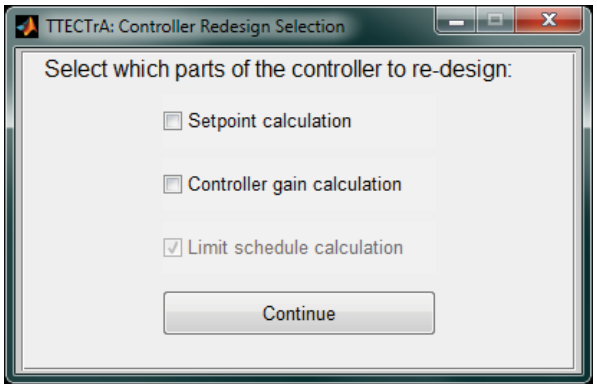

Figure 6.—Redesign Controller GUI.

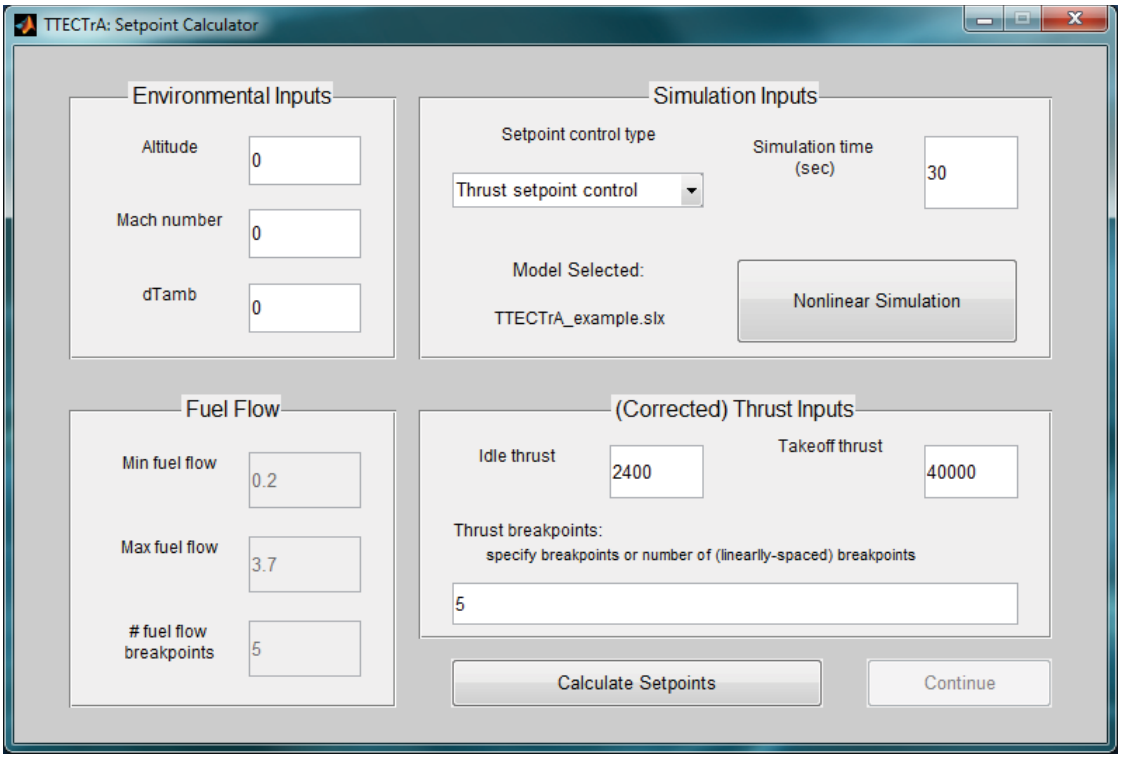

Figure 7.—The Setpoint Calculator GUI.

3.2.1 Setpoint Calculator

The Setpoint Calculator GUI, shown in Figure 7, allows the user to define the flight condition and parameters for the simulations through which the relationship between corrected thrust and the control variable will be defined. The GUI is divided into four sections: Environmental Inputs, Simulation Inputs, Fuel Flow, and Corrected Thrust Inputs.

3.2.1.1 Calculate the Control Variable Setpoints

1. Enter the environmental condition (altitude, Mach number, and *dTamb*) in the "Environmental Inputs" section. The values specified in the input file will be loaded by default, but may be changed in the GUI.
2. Select the setpoint control type to indicate how the relationship between the corrected thrust and the control variable should be derived. The choices are:

a. Thrust setpoint—constant corrected thrust is provided to the model; the setpoint relationship is defined by the steady-state control variable for each corrected thrust
b. Constant Fuel Flow—constant fuel flow is provided to the engine; steady-state corrected thrust and control variable values at each fuel flow are used to define the relationship

3. Specify how long the simulation should run to allow the engine to establish a steady-state condition for the given input ("simulation time").
4. Select the file containing the nonlinear engine model integrated with the *TTECTrA Simulink block*; if the file is not on the MATLAB path, the location of the model will be added to the path definition.
5. Enter the information necessary to calculate the setpoints, as required by the control type selection made in Step 2.

   If "thrust setpoint" control type is selected:
   The "Corrected Thrust Inputs" section is enabled and the following information should be entered:

   a. Idle thrust
   b. Maximum takeoff (or corrected) thrust
   c. The number of thrust breakpoints or a vector containing the specific thrust breakpoints for which control variable setpoints are to be calculated

   If "constant fuel flow" control type is selected:
   The "Fuel Flow" section is enabled and the following information should be entered:

   a. Minimum fuel flow
   b. Maximum fuel flow
   c. The number of breakpoints at which the relationship between control variable and corrected thrust is to be determined

   In both cases, the breakpoint vector will include the specified minimum and maximum values; if a vector of thrust breakpoints is specified, TTECTrA will check whether the minimum and maximum values are included and add them if they are not.

6. Press the "Calculate Setpoints" button to begin the process of determining the relationship between the corrected thrust and controlled variable. Once the lookup table has been defined, a plot of the setpoint relationship will appear, like that shown in Figure 8.
7. Close the figure window, or click the "Continue" button in the GUI, to accept the calculated setpoint relationship and continue with the control design process. Otherwise, the figure window should be left open and new inputs (such as different breakpoint locations) may be entered and the setpoints recalculated by pressing the "Recalculate Setpoints" button (previously the "Calculate Setpoints" button).

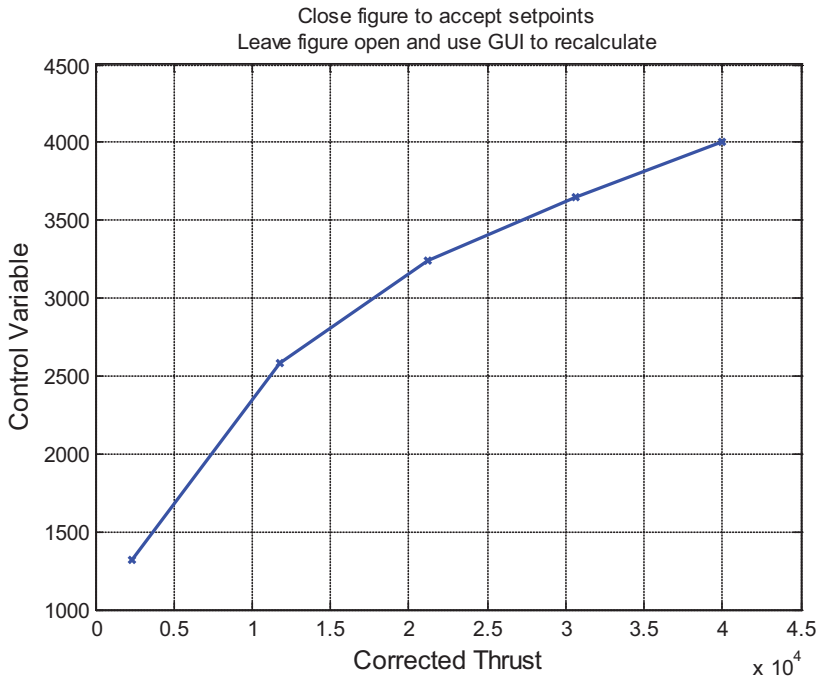

Figure 8.—Example setpoint relationship between the corrected thrust and control variable using TTECTrA.

3.2.2 Control Design Setup

The TTECTrA setpoint controller contains a simple proportional integral (PI) controller with integral wind-up protection, where the PI gains are scheduled as functions of the control variable. The PI gains are found using the MATLAB functions *pidtune* and *pidtuneOptions*, which are included in the Control System Toolbox. The *pidtune* function produces a controller which meets the specifications provided through input arguments to the function and options set using *pidtuneOptions*. In this case, the bandwidth and phase margin of the loop gain (product of the controller and engine model transfer functions) are specified when calling *pidtune*. Prior to designing the gains for the PI controller, TTECTrA allows the user to identify the file containing the linear models used to calculate the gains, and to provide the default tuning parameters, through the Control Design Setup GUI shown in Figure 9. This GUI has two sections: Linear Model Setup and Controller Tuning Setup.

3.2.2.1 Linear Model Setup

To specify the linear model data for control design, the user should specify:

1. The file containing linear model data, which will be shown under the text “Model Selected.” If the correct file is not shown, select the “Load Linear Model” button to open a popup box and browse to and select the correct data (*.mat*) file; if the file is not on the MATLAB path, its location will be added to the path definition. The required format of the linear models in this file is detailed in Appendix B.
2. The index, *i*, of the element in the linear model output vector corresponding to the control variable (i.e. the control variable is the i^{th} element of the output vector y, y_i).

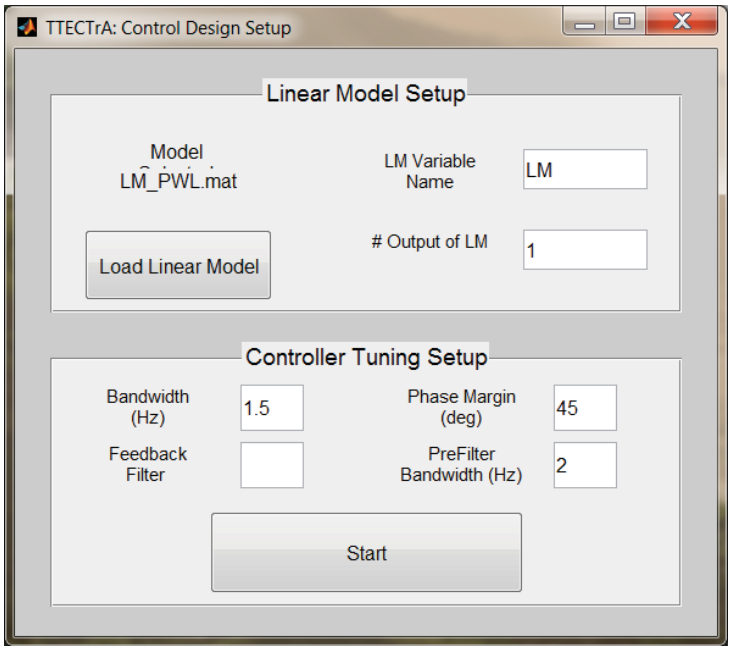

Figure 9.—The TTECTrA Control Design Setup GUI.

3.2.2.2 Controller Tuning Setup

To define the initial parameters for tuning the PI controller gains for each linear model, the user should specify:

1. The desired bandwidth of the loop gain (Hz), which is the product of the controller and plant transfer functions.
2. The desired phase margin (degrees).
3. The bandwidth (Hz) of the feedback filter in Figure 1, which filters the control variable error (difference between setpoint and feedback). The feedback filter will not be used if no value, or a negative value, is provided. If EPR is the control variable, then it is recommended to specify a feedback filter bandwidth of 10 Hz, especially if the engine model is zero-dimensional (does not contain volume dynamics).
4. The bandwidth (Hz) of the prefilter in Figure 1, which filters the thrust command and does not affect the stability of the system.

Once all the data is entered, the user can press the "Start" button to begin control design.

3.2.3 TTECTrA Controller AutoTune

At each thrust point for which a linear model was provided, a PI controller is designed using the design parameters specified in the Controller Tuning Setup section of the Control Design Setup GUI. The window shown in Figure 10 will appear, displaying metrics for the controller designed for a specific linear model. Bode plots of the open loop plant (engine only) and the loop gain (engine and controller) are shown in the leftmost column; the top plot shows the magnitude response and the bottom plot shows the phase response. The top right plot shows the root loci of the plant and loop gain and the bottom right plot shows the step response of the closed-loop linear model with and without a prefilter on the input (labeled "Pre-Filter" and "LM," respectively). This latter information is useful in determining whether modification of the prefilter bandwidth is warranted. The Controller AutoTune GUI, shown in Figure 11, also appears, indicating the thrust point of the model for which the controller is being designed and displaying the rise time and settling time of the step response of the linear model response with a prefilter, along with the bandwidth, phase margin, and proportional (*Kp*) and integral (*Ki*) gains of the PI

controller, in the Current Controller Design section. If the response is not satisfactory, the controller can be retuned (for a specific thrust point) by specifying a different bandwidth and/or phase margin in the Controller Design Inputs section of the GUI. Enter a bandwidth greater than the current bandwidth as the "Bandwidth" input to design a more aggressive controller. Enter a phase margin larger than the current phase margin as the "Phase Margin" input to reduce overshoot in the step response.

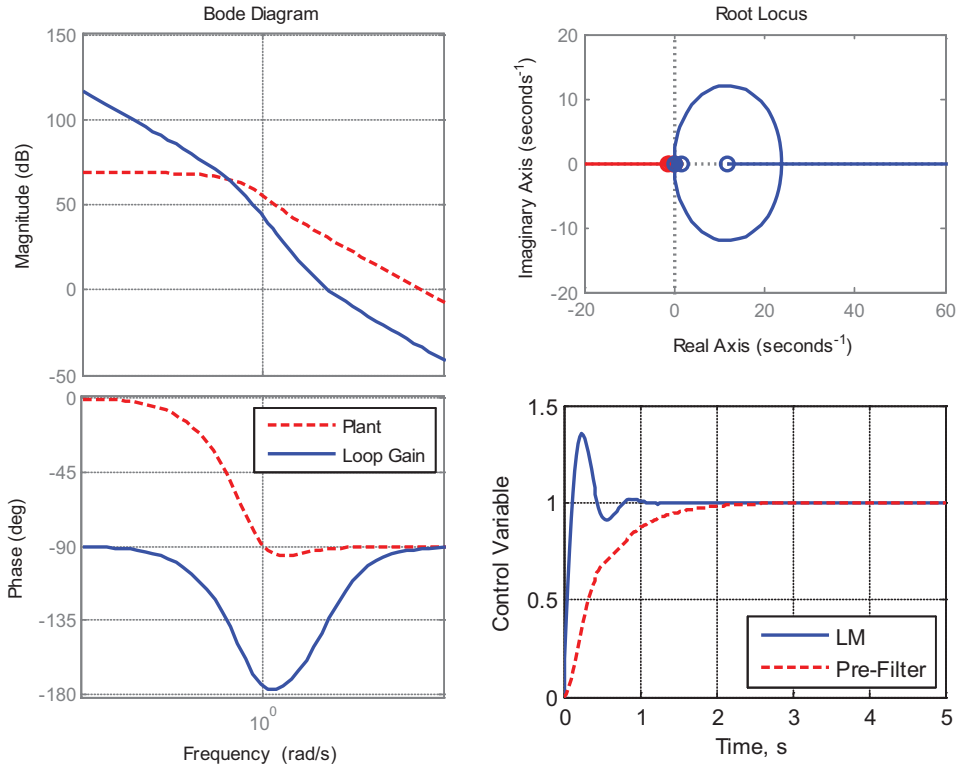

Figure 10.—The controller auto tune output showing the Bode plot, root locus, and linear model response of the current controller.

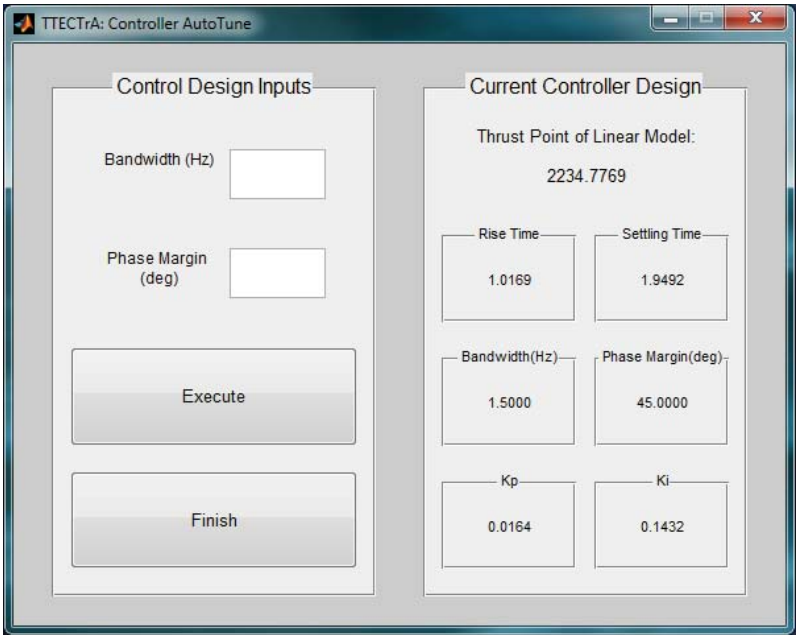

Figure 11.—The Controller AutoTune GUI.

1. Press the “Execute” button to recompute the controller and update the performance metrics (Figure 10).
2. Once the performance is satisfactory (for a given thrust point), press the “Finish” button to design the controller for the next thrust breakpoint.

This process gets repeated for each thrust breakpoint, tuning the controller gains for each breakpoint in the linear model. Once the current breakpoint finishes, the Figure 10 plot will appear for the next point using the bandwidth and phase margin specified in the Controller Tuning Setup step.

3.2.4 Transient Limiter Design

Transient limiters are designed to protect against engine surge during acceleration and deceleration. The Transient Limiter Setup GUI opens after the setpoint controller has been designed, and contains sections for designing the acceleration and deceleration limiters, as shown in Figure 12.

3.2.4.1 Design the Acceleration Limiter

1. Enter the minimum HPC surge margin that is acceptable during an acceleration
2. Press the “Design Accel Limiter” button to begin calculation of the acceleration schedule
3. Once the acceleration schedule is designed, the “Show Accel Schedule” button will be enabled; pressing this button will display the acceleration schedule, as shown in Figure 13. (A default schedule will be displayed if this button is pressed prior to step 2.)
4. To redesign the schedule (for a different minimum surge margin), go back to step 1.

3.2.4.2 Design the Deceleration Limiter

1. Enter the minimum LPC surge margin that is acceptable during a deceleration.
2. Press the “Design Decel Limiter” button to begin calculation of the deceleration limiter.
3. Once the limiter has been designed, the value will be shown under the “Designed Wf/Ps3 Limiter” heading. If the desired minimum surge margin is too low, “NaN” will be shown under “Designed Wf/Ps3 Limiter.” The limiter will show 0, as in Figure 12, if the deceleration limiter has not yet been designed.

After designing both the acceleration and deceleration limiters, the “Continue” button will be enabled; pressing this button will continue running TTECTrA.

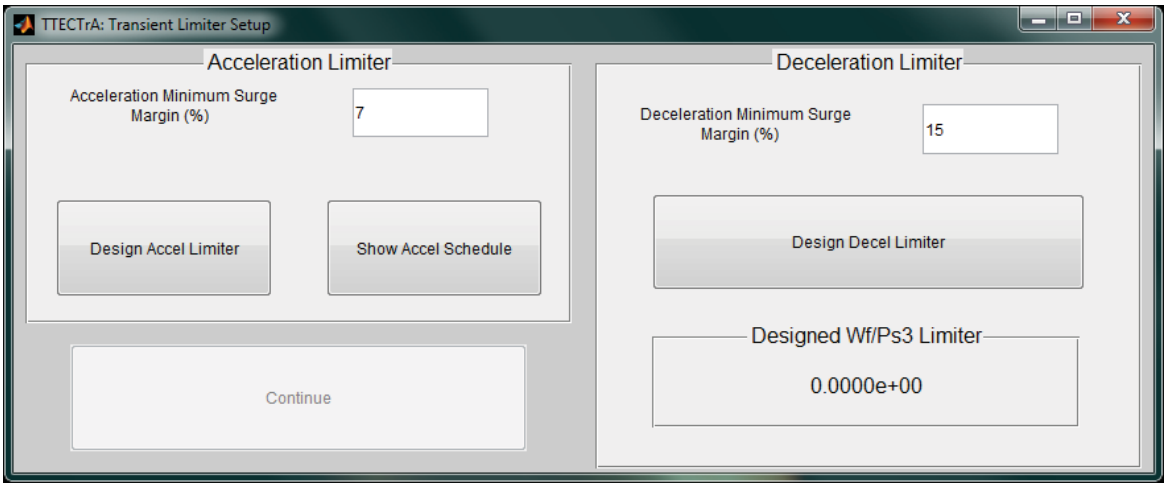

Figure 12.—Transient Limiter Setup GUI.

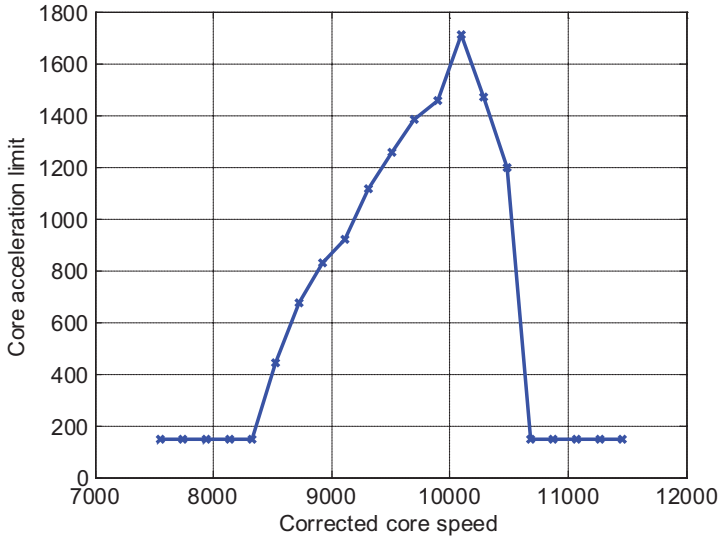

Figure 13.—Example acceleration schedule.

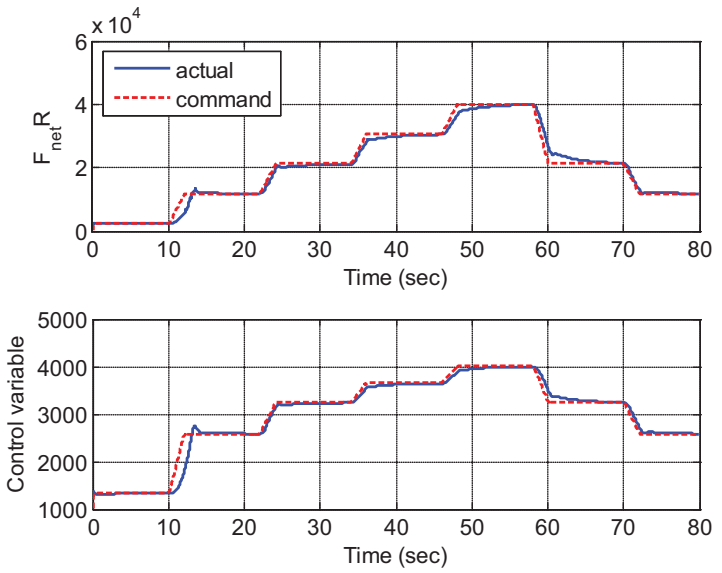

Figure 14.—Thrust and control variable commands and outputs for small thrust transients to test the setpoint controller.

3.2.5 Verification/Simulation

Before running verification simulations, TTECTrA will execute the function *TTECTrA_integration.m* to calculate the integral wind-up protection (IWP) gain for the setpoint controller. This is done automatically and requires no user interaction. The process involves running multiple closed-loop simulations of the model and adjusting the IWP gain to reduce the maximum overshoot below a specified threshold.

The full controller is tested by simulating the model with two different thrust transients, constructed from the control variable setpoints from Figure 7. The first is a series of four small, equally-spaced transients from minimum (idle) to maximum (takeoff) thrust, which tests the setpoint calculator and controller. Figure 14 shows a comparison of actual to demanded thrust and control variable for simulation of the example model with the included controller (top and bottom plots, respectively). The plots show that both the control variable and thrust are driven to the commanded values, even though the controller has no knowledge of the actual thrust produced by the engine.

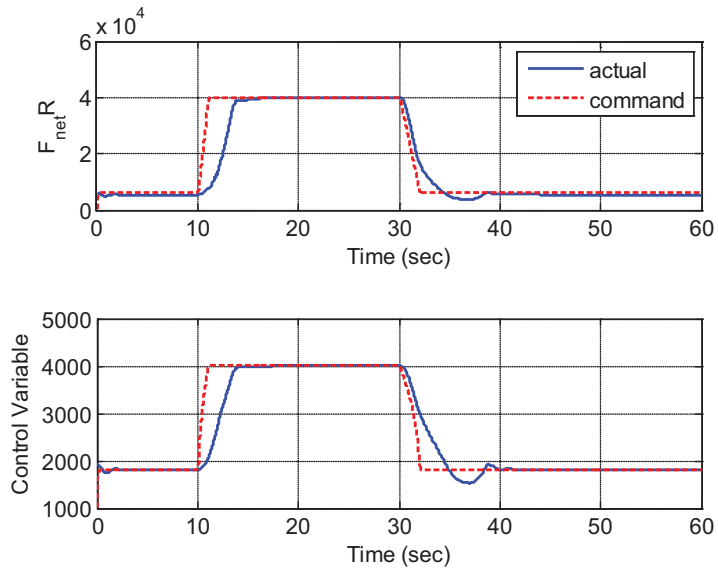

Figure 15.—Actual and commanded thrust and control variable for large thrust transient to test the transient limiters.

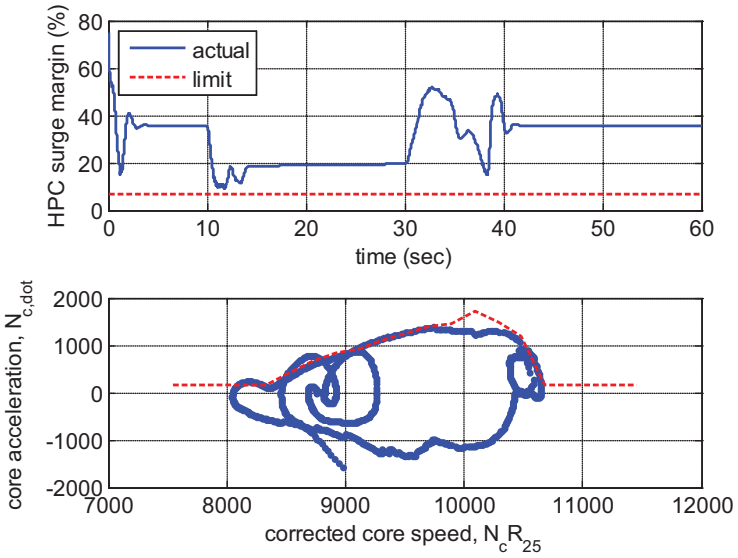

Figure 16.—The HPC surge margin and acceleration schedule for the large thrust transient.

The second transient profile is a large throttle transient, from the largest of 14% of the takeoff thrust and the minimum thrust to the maximum thrust (specified in the control variable setup). The actual and commanded thrust and control variable for the example application are compared in Figure 15. The verification simulations can be changed by modifying the data assigned to the fields ttectra_in.in.t_vec and ttectra_in.in.FT_dmd under the Test Controller Design section of *TTECTrA.m*.

Although the controller is able to drive the engine to both the takeoff and idle thrust values, the presence of the limiters slows this response significantly, as can be seen in Figure 15. When the engine begins to accelerate at low core speeds, it is operating at or near the acceleration schedule limit and the controller restricts the fuel flow to the engine to protect from surge, increasing the time it takes for the engine to reach takeoff thrust from idle thrust. This is reinforced by the top plot of Figure 16, where the HPC surge margin can be seen to approach the limiting value on the same time range in which the responses in Figure 15 are slowed during acceleration. Similar results are seen in Figure 17 during deceleration, where the limit on

Wf/Ps3 is reached, but not exceeded, protecting the engine from violating the LPC surge margin. Like the HPC surge margin, the LPC surge margin remains above the minimum limit (the top plots in Figure 16 and Figure 17), approaching it only at the time in which the limit on *Wf/Ps3* is reached. These results demonstrate that the limiters are working as intended for the controller designed for the example model.

3.2.6 Save Controller Data

Once simulation of the closed-loop model has completed, the dialog box in Figure 18 will appear, asking the user if they would like to save the controller data.

To save the controller data, select "yes" to bring up the Save Data GUI in Figure 19 and enter the name of the .mat file in which to save the data, then press "Save Data." A confirmation message will print to the MATLAB window stating that the file has been saved. Select "no" to skip saving the controller.

The tool will finish running once the controller has been saved (or the user has selected "no" and the window has closed).

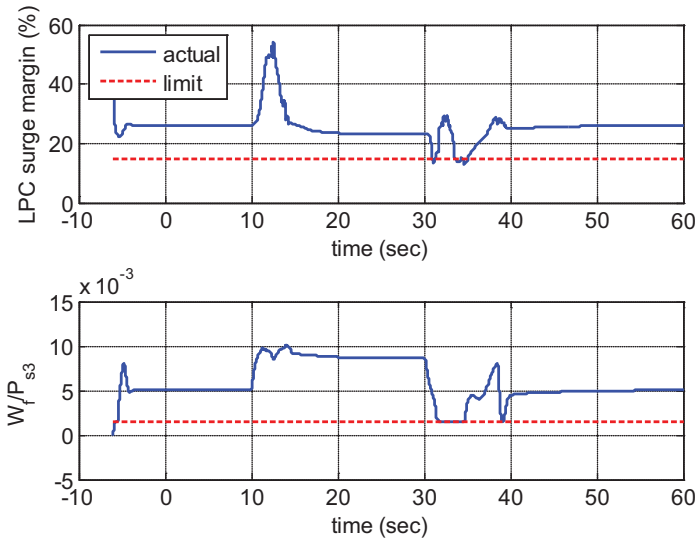

Figure 17.—The LPC surge margin and Wf/Ps3 limit for the large thrust transient.

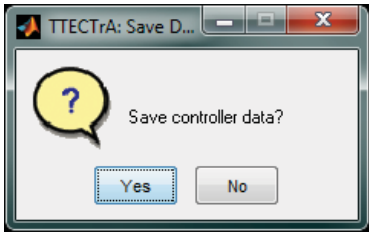

Figure 18.—Save controller popup box.

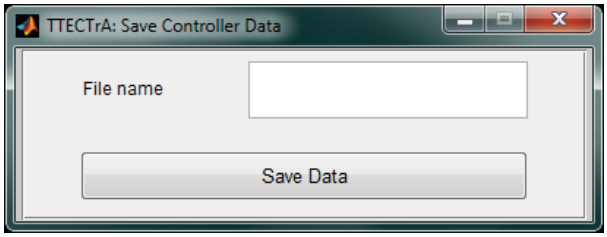

Figure 19.—Save Controller Data GUI.

Appendix A.—Nomenclature

A.1 Variables

accel	Acceleration
alt	Altitude*
decel	Deceleration
dTamb	Free-stream static temperature minus standard atmosphere temperature*
DWS	Dynamic systems analysis workspace (WS) variable
Fdbk	Feedback signal
Fnet	(Uncorrected) thrust*
i	Index for input or output vectors of linear models (*u* and *y*)
inputs	Variable containing inputs for configuring simulation of engine model with TTECTrA
Kp	Controller proportional gains
Ki	Controller integral gains
MN	Mach number
Nc	(Uncorrected) core speed*
Ncdot	Core speed acceleration*
NcR25	Core speed corrected at station 25*
Nf	(Uncorrected) fan speed*
outputs	Variable containing outputs from simulation of engine model with TTECTrA
P2	Pressure at station 2*
Ps3	High-pressure compressor static pressure*
u	Input vector for linear models
Wf/Ps3	Ratio of fuel flow to static high-pressure compressor static pressure
y	Output vector for linear models

* Units of these variables are model-specific

A.2 Acronyms

EPR	Engine Pressure Ratio
GUI	Graphical User Interface
HPC	High-Pressure Compressor
IWP	Integral Wind-up Protection
LM	Linear Model
LPC	Low-Pressure Compressor
PI	Proportional Integral controller
TTECTrA	Tool for Turbine Engine Closed-loop Transient Analysis

Appendix B.—Linear Model Input Requirements

The Tool for Turbine Engine Closed-loop Transient Analysis requires a "good" state space linear model at a minimum of two thrust points. Here, "good" implies that the linear model converges at a given thrust breakpoint so that the four state-space matrices can be found. The linear models should be saved to a structure array that has a length equal to the number of thrust breakpoints for which a steady-state model has been obtained. The structure should contain the following fields:

- A – state matrix of state space model (A)
- B – input matrix of state space model (B)
- C – output matrix of state space model (C)
- D – feed-through matrix of state space model (D)
- IC – Initial conditions or trim values at the linearization point
- Fn – thrust breakpoint at which the model has been constructed

If an automated script is used to determine the linear models, it is possible that some models will not converge; in this case, the matrices and initial conditions (or trim values) can be entered as empty inputs. The controller code will print a warning to the command window and continue with the linear model of the engine at the next thrust point when it encounters an "empty" model.

Appendix C.—Controller Elements to be Verified Against Another Model

The *TTECTrA Simulink Block* has been designed and tested on an in-house engine model and includes the following control elements in addition to those designed by TTECTrA:

- A gain correction on the setpoint controller based on *P2*, primarily to decrease the gain of the controller at higher altitudes.
- IWP gain in the setpoint controller, calculated by running simulations of the closed-loop model to find the gain that reduces overshoot during acceleration and deceleration below a specified threshold
- PI gains in the thrust setpoint controller, designed using linear models of the in-house model, and further adjusted to improve the model response
- PI and IWP gains in the acceleration schedule, designed (using the in-house model) to depend on the ambient pressure (altitude) at which the model is being simulated

Although necessary to obtain acceptable results from simulations of the in-house model, these modifications may not be required when the controller is implemented with other engine models. A piecewise-linear version of this in-house model has been developed and tested successfully with the tool, but it is necessary to test the controller block with other engine models (independent from the in-house model) to verify the necessity of these additional elements.

References

1. Jaw, L., and Mattingly, J.D., *Aircraft Engine Controls: Design, Systems Analysis, and Health Monitoring*, American Institute of Aeronautics and Astronautics, Inc., Virginia, 2009.
2. Mattingly, J.D., Heiser, W.H., and Pratt, D.T., *Aircraft Engine Design*, American Institute of Aeronautics, Inc., 2nd Edition, Virginia 2002.
3. Csank, J., May, R.D., Litt, J.S., and Guo, T.-H., "Control Design for a Generic Commercial Aircraft Engine," AIAA-2010-6629, 46th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit, Nashville, TN, July 25-28, 2010.
4. May, R.D., Csank, J., Lavelle, T.M., Litt, J.S., and Guo, T.-H., "A High-Fidelity Simulation of a Generic Commercial Aircraft Engine and Controller," 46th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit, Nashville, TN, July 25-28, 2010.